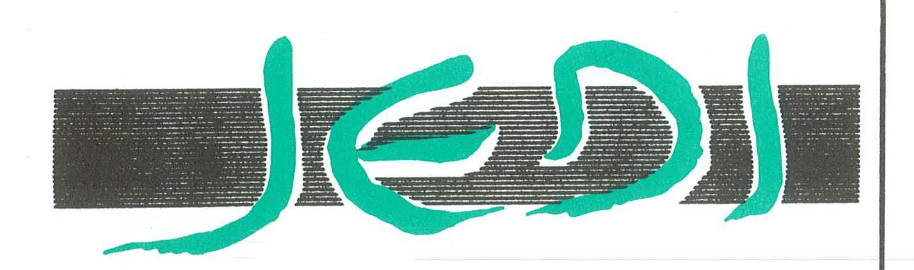


41

JANVIER 1988

LA REVUE QUI N'A PAS BESOIN DE PASSER A L'EST POUR ADHERER AUX PCs ET COMPATIBLES

ET IL Y A ENCORE DES PROGRAMMEURS QUI NE CONNAISSENT PAS TURBO-FORTH...!

EDITORIAL

Ce mois-ci, JEDI fait un doublon: un robot logiciel en LISP et en PROLOG. A vous d'apprécier celui qui vous convient le mieux. JEDI prend des RISC en vous parlant du NOVIX qui n'est maintenant plus une arlésienne: il est disponible en France. Quant à TURBO-Forth, il pousse bien et se fait les dents sur le matériel (carte CGA...) et les routines inter-logicielle: plus et mieux pour les programmeurs qui bougent. Nous attendons vos programmes puisque maintenant vous êtes un certain nombre à avoir acquis la version d'évaluation.

SOMMAIRE

Nos coordonnées: ASSOCIATION JEDI, 17, rue de la Lancette 75012 PARIS
tel président: (1) 43.40.96.53
tel secrétaire: (1) 46.56.33.67

FORTH

NOVIX 4016
QUAND LE LANGAGE MACHINE FAIT UN PAS VERS FORTH

par Marc PETREMANN

Le traitement parrallèle de l'information, on y vient. Le premier micro-ordinateur à technologie RISC vient de sortir en Grande Bretagne: l'ARCHIMEDES d'ACORN. Les langages également s'adaptent: OCCAM et les Transputers sur T800-30. Sommes-nous à un nouveau tournant de la micro-informatique. Peut-être, si l'on considère les performances du traitement parrallèle, pouvez-vous déjà mettre aux oubliettes votre PS 80386 et son OS/2.

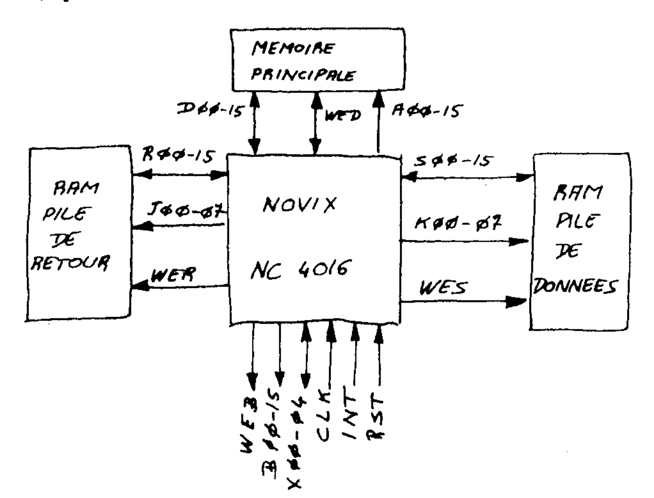
Depuis que la littérature informatique parle de traitement parrallèle, il était temps qu'un micro-processeur exploitant cette technologie apparaisse dans un produit commercialisé. Mais la vraie surprise réside surtout dans le jeu d'instructions de ce composant révolutionnaire, le langage FORTH.

Le micro-processeur NOVIX NC 4016 est construit à partir de la technologie RISC. De nombreux articles ont déjà traité ce sujet, dont MICRO-SYSTEMS récemment. Le terme RISC est l'abréviation de 'Reduced Instruction Set Code'. Un micro-processeur à technologie RISC a moins d'instructions qu'un micro-processeur conventionnel. Pratiquement toutes les instructions internes sont traitées en parrallèle.

Le NC 4016 est constitué de 4000 portes logiques, soit 16000 transistors CMOS, ce qui est peu comparé à un micro-processeur comme le Z80 qui compte pratiquement dix fois plus de portes logiques. La faible consommation électrique du NC 4016 le rend exploitable sur des systèmes portables alimentés par batterie. Sa simplicité architecturale lui permet d'exécuter la majorité des instructions élémentaires en un seul cycle machine, certaines instructions FORTH pouvant être combinées et exécutées en un seul cycle. La finalité du NC 4016 est de réconcilier système et programme en optimisant les sous-programmes.

Le NC 4016 utilise deux piles et deux pointeurs de pile. Toutes les opérations sont exécutées sur la pile de données et le déroulement du programme est contrôlé par la pile de retour. Celle-ci a une capacité de 256 éléments externes et un 257ème élément interne qui est un registre pointant le sommet de la pile de retour. La pile de données externe a une capacité de 256 éléments et les deux éléments figurant au sommet de la pile étant dans les registres T et N, soit 258 éléments au total.

Avec le NC 4016, la mémoire est adressée par mots et non par octets, un Kmots représente deux Koctets. L'accès à la mémoire est assuré par un vrai bus de données 16 bits et s'étend sur 64 Kmots (128 Koctets). Cet espace mémoire peut être étendu en utilisant le port 5 bits X comme sélecteur de page mémoire, la capacité totale pouvant alors atteindre 32 pages de 64 Kmots (soit 4 Moctets).



Les deux ports d'entrée-sortie B et X sont contrôlés par un jeu de quatre registres:

- registre de données
- registre de masquage
- registre de direction
- registre de contrôle à trois états (Tri-state Control)

Chaque bus, mémoire principale, pile de données, pile de retour, et ports d'entrée-sortie, peut être activé de manière indépendante à chaque cycle d'horloge.

Les appels et les retours d'une définition de type deux-points ne prennent qu'un ou deux cycles d'horloge, ainsi les programmes modulaires sont exécutés très rapidement. Les nombres compris entre -1 et 32 sont générés en un seul cycle, tous les autres nombres 16 bits signés le sont en deux cycles. L'appel d'un sous-programme ne prend qu'un cycle d'horloge. Le retour de sous-programme ne requiert aucun cycle d'horloge.

Les registres internes du NC 4016 sont nombreux et permettent un contrôle optimal des données placées sur la pile, en mémoire ou vers les ports d'entrée-sortie:

T	Elément du sommet de la pile de données
N	Second élément de la pile de données
Nc	Bit de retenue du second élément de la pile
J/K	pointeur combiné de pile de données(K) ou retour(J)
I	Elément situé au sommet de la pile de retour
P	Compteur programme, lecture seulement
-1	booléen "vrai", lecture seulement
MD	Multiplication/Division, registre d'utilisation temporaire
SR	Racine carrée, registre d'utilisation temporaire
Bport	Registre de données du port B
B-Mask	Registre de masquage, 1 = invalide
B-I/O	Registre de direction, 1 = sortie
B-TRI	Mode trois états, 1 = trois états en écriture
Xport	Registre de données du port X
X-Mask	Registre de masquage, 1 = invalide
X-I/O	Registre de direction, 1 = sortie
X-TRI	Mode trois états, 1 = trois états en écriture
TIMES	Identique au registre I, mais contrôle la répétition de la dernière instruction avec auto-décrémentation de son contenu

Le port B sert essentiellement à communiquer avec un environnement matériel tel que cartes d'extensions (communication asynchrone) ou des périphériques d'acquisition de données (convertisseur A/N N/A, souris, crayon optique, synthèse vocale, etc...).

CODAGE DES INSTRUCTIONS DU NC 4016

Les instructions exécutables par le micro-processeur NC 4016 sont classées en huit catégories. La catégorie d'une instruction dépend de la valeur des quatres bits de poids fort d'une instruction:

fonc- bit tion	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
call	0	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
arithmét.	1	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x
IF	1	0	0	1	a	a	a	a	a	a	a	a	a	a	a	a
LOOP	1	0	1	0	a	a	a	a	a	a	a	a	a	a	a	a
ELSE	1	0	1	1	a	a	a	a	a	a	a	a	a	a	a	a
litéral@	1	1	0	0	x	x	x	x	x	x	l	l	l	l	l	l
litéral!	1	1	0	1	x	x	x	x	x	x	l	l	l	l	l	l
mémoire @	1	1	1	0	x	x	x	x	x	x	l	l	l	l	l	l
mémoire !	1	1	1	1	x	x	x	x	x	x	l	l	l	l	l	l

a: bits d'adresse x: bits de contrôle l: bits litéraux

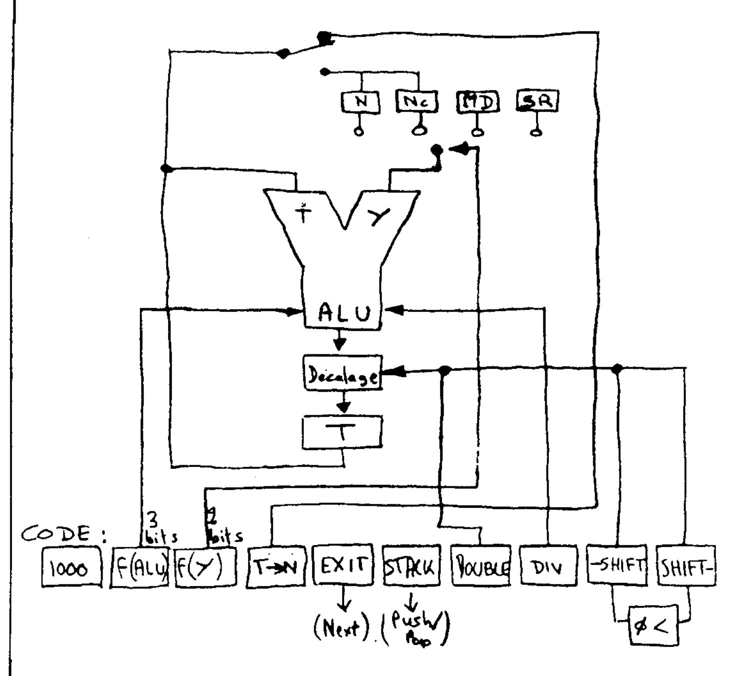
- L'instruction CALL est implicite. Elle est indiquée par la mise à zéro du bit de poids le plus fort du contenu d'une cellule 16 bits et exécute le sous-programme commençant à la position indiquée par les bits b0-b14. Cette valeur est à multiplier par deux pour obtenir une adresse physique réelle, un sous-programme commençant

toujours à une adresse paire. Exemple:

```
3140h   signifie CALL 3140h
( exécution du s-prg commençant à la cellule 3140h)
( cellule 3140h correspond à adresse physique 6280h)
```

- Les instructions arithmétiques sont codifiées à partir du contenu des bits b9-b11 pour le code de l'opération à exécuter par l'unité arithmétique et logique (ALU) et les bits b7-b8 pour la sélection des registres d'entrée-sortie.

L'unité arithmétique et logique réalise en plus des opérations d'addition et de soustraction, les opérations de multiplication, de division et d'extraction de racine carrée sur 16 bits. Le principal registre source est T pour l'élément figurant au sommet de la pile de données. Le registre source secondaire est sélectionné parmis les registres N, MD ou SR en fonction des opérations à exécuter. Le résultat de l'opération est chargé dans le registre T, c'est à dire déposé au sommet de la pile de données.



Détail de f(ALU) indiqué par b11-b9

bits 11	10	9	Opération
0	0	0	passe T
0	0	1	T AND Y
0	1	0	T - Y
0	1	1	T OR Y
1	0	0	T + Y
1	0	1	T XOR Y
1	1	0	Y - T
1	1	1	passe N

Détail de f(Y), entrées en T et Y, sortie en T, indiqué par b8-b7

bits 8	7	Registre sélectionné
0	0	N
0	1	Nc
1	0	MD
1	1	SR

Exemple de codage de trois instructions élémentaires:

```
 +   1000 1000 0001 0000   soit 8810h
 -   1000 1100 0001 0000   soit 8C10h
OR   1000 0110 0001 0000   soit 8610h
```

L'instruction dite "d'assemblage" chargée de compiler une fonction sera de la forme:

```
 : 1MI  CREATE , DOES> @ , ;
HEX  8810 1MI +,
     8C10 1MI -,
     8610 1MI OR,  DECIMAL
```

Cette définition est reprise du F83 accompagnant la carte NB1000 pour IBM PC et compatibles et équipée du NC4016.

- Les instructions de branchement IF, ELSE et de boucle LOOP indiquent un branchement sur les huits octets de poids faible vers la cellule correspondant à cette valeur. Le déplacement possible est situé dans l'intervalle-2048d..2047d, valeurs indiquant un déplacement en nombre de cellules (paires d'octets).

- Les valeurs litérales sont déclarées à l'aide d'une combinaison de bits dont les bits b0-b4 indiquent la valeur à déposer sur la pile. Toutes les valeurs littérales comprises entre 0 et 31 peuvent être déclarées et traitées en un seul cycle d'horloge. Les valeurs littérales supérieures à 32 sont traitées par empilage du contenu d'une cellule, excepté la valeur -1 qui est extraite d'un registre spécialement affecté à cette valeur. Ainsi, le FORTH du NC4016 n'a pas besoin de définir les constantes TRUE, FALSE, 0, 1, 2 et 3 qui existent dans le F83 MSDOS. Exemple, la compilation de:

```
5 7 +
```

s'exécute en trois cycles d'horloges et n'occupe que trois mots (6 octets) dans la mémoire du NC 4016!!

Liste des 40 primitives FORTH du NC 4016:

ACCES MEMOIRE:

```
 @    empile une valeur dont l'adresse est pointée par le
      sommet de la pile
 !    range en mémoire une valeur dont l'adresse est au
      sommet de la pile
nn@   empile depuis un pseudo-registre
nn!   range dans un pseudo-registre
I@    empile depuis un registre interne
I!    range dans un registre interne
n@    empile depuis l'adresse pointée par l'adresse 16
      bits qui suit cette instruction.
nn@   empile un litéral 5 bits (pseudo-constante)
nn!   range un litéral 5 bits
```

ARITHMETIQUE ET LOGIQUE:

```
 +    addition en complément à deux
 +c   addition avec retenue pour traitement de nombres
      32, 48, 64 bits ou plus
 -    soustraction en complément à deux
 -c   soustraction avec retenue
OR    ou inclusif logique bit à bit
AND   et logique bit à bit
XOR   ou exclusif logique bit à bit
 2/   décalage arithmétique à droite
 2*   décalage arithmétique à gauche
 0<   remplace le nombre située au sommet de la pile par
      flag booléen vrai si nbr < 0
D2/   décalage arithmétique 32 bits à droite
D2*   décalage arithmétique 32 bits à gauche
*'    pas de multiplication
*-    multiplication signée
*F    multiplication fractionnaire
/'    pas de division
/"    dernier pas de division
S'    pas d'extraction de racine carrée
```

STRUCTURES DE CONTROLE:

```
if     saut conditionnel si sommet de pile est nul
else   saut inconditionnel
#loop  saut avec décrémentation de compteur si pas nul
times  initialise le compteur de répétition d'instruc-
       tion; nbr répétition prélevé sur sommet de pile
call   appel sous-programme (optimisé à 1 cycle machine)
exit   retour de sous-programme
```

MANIPULATION DE LA PILE:

```
dup   copie du sommet de la pile
drop  dépile le contenu du sommet de la pile
r>    transfert pile de retour vers pile de données
```

```
  r@    copie pile de retour vers pile de données
  #I    copie index de boucle au sommet de pile de données
  >r    transfert pile de données vers pile de retour
```

Certaines instructions peuvent être combinées en un seul code, exemple:

```
  @+  @-      @+c   @-c
  @or   @xor    @and
```

Le nombre de combinaisons de primitives est de 123. La liste détaillée de ces instructions est disponible dans la documentation du NC 4016.

UTILISATION PRATIQUE DU NC 4016: LA CARTE NB 4100

Afin de réaliser un article objectif, nous nous sommes fait prêter par la société M.I.E.L. une carte NB 4100 équipée du NC 4016. Je remercie au passage Mr RENAUDIN pour son aimable collaboration.

La carte NB 4100 est une carte d'extension au format IBM enfichable sur un des connecteurs d'extension de la carte mère du système IBM PC ou compatible. La carte est insérée bien évidemment lorsque le système hôte est hors tension. Les essais de la carte ont été réalisés sur deux systèmes différents: un TOTO équipé de 640 K de RAM, un disque dur 20 Moctets; un second système BONDWELL PC avec seulement 256 K de RAM, deux lecteurs de disquette. La mise en route n'a posé aucune difficulté sur ces deux systèmes.

La carte NB 4100 est équipée, outre son micro-processeur NC 4016, d'une mémoire RAM de 64 Kmots (128 Koctets), un espace RAM supplémentaire de 8 Kmots est utilisée pour les piles de données et de retour et permet le traitement multi-tâches jusqu'à 32 tâches indépendantes. La configuration de cette carte permet de reprendre la main sur le système hôte tout en laissant le NC 4016 continuer le traitement en cours.

Deux manuels accompagnent la carte:

- NOVIX USER'S GUIDE: guide d'utilisation et détails techniques concernant la carte NB 4100 et du NC 4016
- NOVIX EXPRESS REFERENCE MANUAL: manuel de référence du vocabulaire FORTH 83-Standard et de l'éditeur de bloc plein écran NOVIX EXPRESS. Ce manuel est complété par une adaptation de l'ouvrage de C.H. TING 'INSIDE F83'.

On ne peut que regretter le manque de détails documentaires concernant le NC 4016. Un vrai TUTORIAL aurait été souhaitable avec un produit aussi performant. La documentation en l'état n'est abordable que par un professionnel de l'informatique et disposant d'une certaine pratique de F83.

Une disquette 5'1/4 au format IBM PC (360K) contient le système d'exploitation NOVIX, le langage FORTH, le module de communication, des utilitaires divers, un méta-générateur et les principaux fichiers sources.

La communication entre le système IBM PC et la carte NB 4100 est activée en lançant le programme TALK4100.COM fourni avec la disquette. Ce programme contient le système d'exploitation propre à la carte NB 4100. La communication est réalisée par utilisation des mots PC!, P!, PC@ et P@ normalement implantés dans le F83 de Laxen et Perry.

Un système de menu par touches de fonctions permet de sélectionner diverses options telles que:

- F1 affiche un menu d'aide.
- F2 réinitialise la carte NB 4100.
- F3 est similaire à F2, mais active après initialisation un programme précompilé disponible dans le fichier KNRL4100.N41. Ce programme provient de la méta-génération d'un programme défini par l'utilisateur.
- F4 bascule de contrôle MS DOS-NOVIX.
- F5 définit et ouvre un fichier de sauvegarde d'une session de travail.
- F6 permet le chargement et l'exécution d'une suite d'instructions préalablement sauvegardée dans un fichier. Cette interprétation s'interrompt en fin de fichier.
- F7 permet d'utiliser un code de contrôle sans l'interpréter.
- F8 sauvegarde une séquence de frappe de commandes au clavier dans un fichier. Ce fichier peut être réutilisé par F6.
- F9 lance l'éditeur de fichiers NOVIX.
- F10 quitte le programme TALK4100 et rend la main à MSDOS.

Pour donner le contrôle du système au NC 4016, on active la fonction F2. Le programme de communication MSDOS-NOVIX est chargé. Ensuite, l'appui sur F4 bascule le contrôle du moniteur de MSDOS au NOVIX. Pour confirmation, un appui sur la touche RETURN affiche:

```
  Novix ok>
```

Vous pouvez commencer à travailler directement avec le NC 4016. Tout ordre tapé au clavier est interprété ou compilé directement par le programme situé dans la mémoire vive de la carte NB 4100 et sous contrôle du NC 4016.

Pour conserver une trace des manipulations effectuées, on active F4 puis F5. On précise le nom du fichier destiné à recevoir les caractères affichées et on rappuie sur F4. A partir de maintenant, toute exécution d'instruction provoquant un affichage sera conservée pour analyse éventuelle dans le fichier texte créé. Exemple de session de travail:

```
  hex Novix ok (HEX) >
  see dism
    4D56       1745      call    1745 CR
    4D57       8050      DUP
  ...etc...
    4D5E       9056      (IF)    4D56
    4D5F       8E30      DROP RETURN
  Novix ok (HEX) >
```

Les programmes sources sont écrits sous forme de blocs de 16 lignes de 64 caractères. Une instruction INCLUDE provoque le chargement et la compilation d'un fichier de blocs. Un fichier source peut être compilé depuis un autre fichier par une ligne INCLUDE intégrée au fichier appelant. Le FORTH NOVIX accepte également d'exécuter des commandes provenant d'un fichier ASCII, un peu à la manière de TURBO-Forth.

La syntaxe du langage FORTH utilisé par la carte NB 4100 est celle définie par le Standard 83 et la version FORTH est une adaptation de celle écrite par Laxen et Perry, la plus utilisée à ce jour. Dans cette version, une primitive FORTH est définie à partir de l'assembleur NOVIX:

```
  CODE DROP   ( n1 -- )   drop,  exit!  END-CODE
  CODE DUP     ( n1 -- n1 n1 )   dup,  exit!  END-CODE
  CODE SWAP    ( n1 n2 -- n2 n1 )   swap,  exit!  END-CODE
  CODE OVER     ( n1  n2 --  n1 n2 n1 )   over,  exit!
END-CODE
```

Voici les définitions équivalentes définies dans le programme source du langage FORTH destiné au micro-processeur 8086:

```
  CODE DROP   (S n1 -- )   AX POP   NEXT END-CODE
  CODE DUP    (S n1 -- n1 n1 )   AX POP   AX PUSH    1PUSH
END-CODE
  CODE SWAP     (S n1  n2 --  n2 n1  )     DX POP   AX POP
2PUSH END-CODE
  CODE OVER    (S n1 n2 -- n1 n2 n1 )
     DX POP   AX POP   AX PUSH   2PUSH END-CODE
```

La carte NB 4100 est exploitable en plusieurs phases:

- la phase de développement qui consiste à définir et tester ses définitions à partir d'un fichier. Cette phase est également celle de l'apprentissage pour ceux qui n'ont pas une pratique intensive du langage FORTH.

- La phase d'optimisation et de mise au point. C'est l'habillage du programme. On y soigne l'ergonomie et effectue le jeu d'essais habituel et l'optimisation des

routines dont le temps d'exécution est critique. On soigne également la présentation et la documentation.

- la phase de méta-génération. En passant par le méta-générateur, on recrée un programme compilé sans s'encombrer des utilitaires de mise au point. La maîtrise de la méta-génération permet d'obtenir des programmes directement exécutables par le système NOVIX et très compacts donc peu encombrants en espace mémoire. Le manque de précision concernant cette troisième étape dans la documentation nous a obligé à de nombreux essais avant de réussir une méta-compilation sans erreur. Nous n'avons pas eu le temps de tester une métacompilation d'application utilisateur.

Le temps de compilation d'un bloc provenant d'un fichier source est très court, mais la compilation d'un fichier complet est handicapée par le temps d'accès au disque, car gérée par interruption et appel d'une fonction BIOS. L'emploi d'un disque dur est presque obligatoire pour développer avec aisance une application consistante.

Pour bien maîtriser le fonctionnement de la carte NB 4100, il est nécessaire d'avoir une certaine pratique du langage FORTH. La syntaxe de ce langage, soi-disant "hermétique" par ceux qui n'ont fait qu'aborder les premiers opérateurs, ceux de manipulation de la pile et la notation polonaise inverse, permet d'aborder des problèmes logiciels dits "de bas niveau comme l'assemblage, ou des problèmes "de haut niveau" comme ceux traités par les langages évolués et structurés (PASCAL, C, MODULA-2...). Dans le cas du NC 4016, les formes "bas niveau" et "haut niveau" s'estompent au profit d'une performance inégalable sur cette gamme de matériel.

La carte NB 4100 est destinée au développement d'applications nécessitant des temps de traitement les plus courts possibles. Un des avantages de la carte NB 4100 est de permettre le déroulement d'un programme sur la carte puis de revenir sous MSDOS et de poursuivre un autre travail sans interrompre celui de la carte. Il n'y a pas de collision dans la gestion des espaces mémoire du DOS ou de la carte, car ces espaces mémoire sont indépendants.

LES PERFORMANCES DU NC 4016

Les tests effectués à partir de la carte NB 4100 sont significatifs de la puissance du NC 4016 et rivalisent avec ceux effectués sur des systèmes dits "mini-ordinateurs". Le premier test est celui de la boucle à vide:

```
: BOUCLE 1000 0 DO 1000 0 DO LOOP LOOP ;
```

qui s'exécute un million d'itérations en moins de 2/10e de secondes. Pour indication, une boucle similaire n'exécutant que 10000 itérations en FORTH 83-Standard sous MSDOS met 3/10e de secondes.

Pour avoir une meilleure idée des temps d'exécution, voici un tableau comparatif des performances d'instructions type entre trois micro-processeurs:

Temps d'exécution comparatifs
entre NC 4016 et deux micro-processeurs standards

OPERATION	NC 4016	8086	68000
Mouvement reg. à reg.	1	2	8
Mouvement reg.vers mém.	2	8-11	18
Multiplication	23	118-133	74
Division	31	144-162	144-162
Appel de sous-programme	1	19-28	32
Retour sous-programme	1-0	8-16	32
Branchement	1	4-16	18
Empilage ou dépilage	1-0	8-10	16

En règle générale, une définition FORTH traitée par le NC 4016 sera plus rapide qu'un programme équivalent assemblé sur un système équipé d'un 8086 ou d'un 68000. Ces performances doivent être tempérées si les programmes font appel à des routines incorporées au DOS du système hôte, comme l'ouverture d'un fichier ou l'affichage d'un caractère.

Le programme d'essai classique pour tester les performances d'un langage (et non d'un système) est le Crible d'Eratosthène dont voici le listing en FORTH tel qu'il est exécuté par le NC 4016 après compilation:

```
\ Programme de détermination des nombres premiers
\ situées entre 0 et 8190:
\  CRIBLE D'ERATOSTHENES
DECIMAL  8190 CONSTANT CLIMIT
VARIABLE COMPTE VARIABLE ITER VARIABLE IVAL
VARIABLE TABLE CLIMIT ALLOT
: CRIBLINIT  1 COMPTE !   TABLE CLIMIT TRUE FILL ;
: ELIMINE
   BEGIN  ITER @ CLIMIT > NOT
          IF  0 TABLE ITER @ + C! THEN
          ITER @ IVAL @ + DUP ITER !  CLIMIT >   UNTIL ;
: CRIBLE
    CLIMIT 2  DO   TABLE I + C@
             IF   I DUP DUP IVAL ! + ITER !
             ELIMINE 1 COMPTE +!   THEN    LOOP ;
: DOCRIBLE
    CR CR CR CR  25 SPACES ." CRIBLE D'ERATOSTHENES " CR
    25 SPACES ."     10 ITERATIONS " CR  25 SPACES
    ." CALCULS EN COURS .... " CR CR
    9 FOR CRIBLINIT CRIBLE NEXT
    21 SPACES COMPTE @ .
    ." NOMBRES PREMIERS SUR " CLIMIT . CR CR CR CR ;
```

L'instruction FOR est spécifique au FORTH NOVIX mais peut être définie simplement en F83 (Laxen et Perry ou TURBO):

```
: FOR  ( n ---)
0 [COMPILE] DO ; IMMEDIATE
: NEXT ( ---)
[COMPILE] LOOP ; IMMEDIATE
```

TEST D'EXECUTION DU CRIBLE
D'ERATOSTHENES

Système	Langage	Temps en ms
8086/ 5Mhz	Forth	6400
68000/ 8Mhz	Forth	2700
PDP 11/73	Forth	1485
VAX 780	C	142
NC 4016/ 8Mhz	Forth	117
Sperry 1100/82	Fortran	67
Cray-1	Fortran	11

On constate avec stupeur que le NC 4016 est plus rapide qu'un VAX et à peine dix fois plus lent qu'un CRAY-1. En fait, le rapport prix/performances est incomparable.

LE DEVELOPPEMENT LOGICIEL

Nous ne savons pas encore grand chose des développements logiciels liés au NC 4016. Cependant, il existe déjà un Tiny-C écrit en FORTH NOVIX permettant à un programmeur non initié au FORTH d'exploiter le NC 4016. Ce Tiny-C est fourni avec un des produits exploitant le NC 4016.

LES PROCHAINS PRODUITS

La firme NOVIX Inc a en chantier le NC 6016 dont les performances dépassent celles de l'actuel NC 4016. Dès que nous aurons des précisions sur cette nouvelle puce, vous en serez informé.

La généralisation envisageable à court terme de la technologie RISC mettra un terme à la bataille engagée par des produits soi-disant 'nouveaux' (80386 notamment) et verra l'apparition de systèmes avec l'architecture des micro-ordinateurs et les performances des minis. Le développement le plus spectaculaire se produira certainement dans l'infographie et la génération de

systèmes vidéo: palettes graphiques, digitalisation d'images, numérisation vidéo en temps réel - vers le magnétoscope numérique...-, synthèse d'image, synthèse sonore, systèmes experts à hautes performances, etc....

LES PRODUITS EQUIPES DU NC 4016

Actuellement, cinq produits équipés du NC 4106 sont importés et commercialisés en France:

1) La carte NB 4200

Carte de développement utilisable par liaison série RS232 aux vitesses de 9600, 19200 ou 28800 bauds. Est équipée d'une version cmFORTH sur EPROM. Espace mémoire initial de 16koctets RAM, 16koctets EPROM, 4koctets RAM pour les piles de données et de retour.

Des connecteurs amovibles permettent l'extension de l'espace mémoire principal par l'utilisateur, et chaque espace mémoire de pile jusqu'à 128Koctets. Conçu pour une utilisation simple et performante, le NC 4016 opère sur quatre espaces mémoire simultanément et travaille entre 8 et 10 Mips.

Tout système IBM PC ou compatible équipé d'un interface série peut être adapté à la carte. Le système hôte est alors utilisé comme terminal d'édition, de stockage et de chargement des programmes source. Le programme NOVIX.COM fourni sur disquette assure l'émulation du système hôte comme terminal via l'interface COM1. La vitesse de traitement dépendra du temps requis par la mémoire principale pour transférer les données entre le système hôte et la carte NB 4200.

2) La carte NB 4300

Carte d'application professionnelle équipée du bus STD, d'un port parrallèle 16 bits et d'une liaison série permettant la connexion d'un terminal ou d'une imprimante série. Elle peut être associée à d'autres cartes du même type comme carte maîtresse du système ou carte esclave dans un environnement multi-processeur.

La mémoire RAM système occupe 4Kmots à partir de l'adresse zéro. Le système d'exploitation en ROM occupe 8Kmots à partir de l'adresse 1000h. Cette zone contient le noyau FORTH. La mémoire RAM affectée au dictionnaire occupe 28Kmots entre les adresses 2000h et 7FFFh.

Cette carte est livrée avec le système d'exploitation NOVIX EXPRESS. Le compilateur combine automatiquement le cas échéant les instructions exécutables simultanément, comme "OVER SWAP -" par exemple, en une seule instruction.

Différentes configurations sont possibles pour l'utilisation de cette carte:

- intégrée à un système utilisant la carte NB 4300 et pilotant d'autres cartes via le bus STD, à l'exception de cartes d'extensions RAM nécessitant un rafraîchissement périodique.

- une carte NB 4300 maîtresse associée à plusieurs cartes NB 4300 esclaves communiquant par le bus STD.

3) La carte NB 4100

Carte à insérer directement au bus d'extension du système IBM PC ou compatible. Cette carte dispose de sa propre extension mémoire et le NC 4016 travaille indépendamment du micro-processeur équipant le système hôte.

Cette carte dispose d'una capacité RAM de 64Kmots intégralement accessibles par le NC 4016. Une zone RAM de 8Kmots indépendante est utilisée pour les piles de données et de retour et permet le traitement multi-tâches jusqu'à 32 tâches indépendantes. En outre, la configuration de cette carte permet de reprendre la main sur le système hôte tout en laissant le NC 4016 continuer le traitement en cours.

4) La carte NB 4000

Carte de développement complète à connecter en série via le port COM1 à un système IBM PC ou compatible. Fonctionne à l'aide du logiciel polyFORTH. Cette carte est orientée vers les applications multi-tâches, génération de signaux, acquisition de données, etc...

Le logiciel intégré et très compact optimise automatiquement votre programme en combinant les mots pouvant être rassemblés une seule instruction.

Le système occupe 4 Kmots en RAM, le système d'exploitation 4 Kmots en ROM. Le dictionnaire dispose de 24 Kmots en RAM entre les adresses 2000h et 7FFFh.

5) Station autonome de développement ND 4000

Ce système intègre la carte NB 4000, un lecteur de disquette 5'1/4 360k, un disque dur 10 Moctets et une alimentation électrique de 65 Watts. Le tout est livré dans un coffret métallique. Le système se raccorde à n'importe quel type de terminal par l'intermédiaire d'une liaison RS232.

L'environnement de développement polyFORTH complet inclus notamment un compilateur optimiseur, la gestion multi-tâches, interpréteur FORTH interactif, un éditeur, méta-compilateur, extension mathématique avancée.

Deux jeux de connecteurs donnent accès au bus d'adresse 16 bits, bus de données 16 bits, bus 8 16 bits, bus X 5 bits, bus de périphérique lent 16 bits, toutes les lignes d'horloge système et de validation.

REFERENCES:

IMPORTATEUR EXCLUSIF:
Sté MIEL 60, rue de Wattignies 75012 PARIS
Tel(1) 43.42.92.07 Telex: 213005
Demander Mr RENAUDIN

BIBLIOGRAPHIE:

Articles: STACK MACHINES AND COMPILER DESIGN par D.L. MILLER BYTE magazine Avril 87

FORTSCHRITTE BEI FORTH par Dr H.J. GANSER MC-HARD novembre 87

FORTH IN SILIZIUM par P. GLASMACHER C'T cahier 4 1987

Livres: MORE ON NC4000 Vol I à IV

FOOTSPEPS IN AN EMPTY VALLEY par C.H. TING

MANUEL F83-Standard pour systèmes CP/M et MSDOS par M.PETREMANN, J.M.PREMESNIL et M.ZUPAN Ed: LOISITECH

Logiciels: FORTH 83-Standard "Public Domain" version Laxen et Perry

NOVIX EXPRESS SYSTEM

TURBO-Forth 83 Standard ASSOCIATION JEDI

PRIX UNITAIRES HT DES PRODUITS EQUIPES DU NC 4016:

NB 4100	11490 Fr
NB 4200	5590 Fr
NB 4300	9950 Fr
NB 4000	16600 Fr
ND 4000	25120 Fr

Prix carte + logiciel (le cas échéant).

Pour les adhérents de JEDI intéressés par un de ces produits, prendre contact avec le secrétaire pour monter un achat groupé et bénéficier d'une remise auprès de la Sté MIEL.

FORTH

COMMUTATION CARTE MONO - CARTE CGA

par Marc PETREMANN

pour TURBO-Forth

Ma vocation première était, avant l'informatique, le dessin. Et bien entendu, je ne manque jamais d'appliquer la formule: "un bon dessin vaut mieux qu'un long discours". Comme j'ai le crayon paresseux, je préfère faire exécuter les croquis et les schémas par mon ordinateur. Mais on ne passe pas d'un TO7 (fort bon au demeurant s'il est équipé de COLORPAINT) à un compatible IBM strictement minimal.

Et comme toujours, quand on veut tirer plus de possibilités de son micro, on fait l'acquisition de cartes d'extensions. Je me suis donc fendu d'une carte CGA (pour 390 Fr chez AZ COMPUTER, 99 rue Balard dans le 15 arrd, c'est presque donné).

Hélas, les ennuis commencent quand je veux essayer cette merveille de technologie asiatique toute garnie de puces et d'idéogrammes à la signification totalement hermétique: RIEN! Coup de téléphone, passage chez AZ, essai sur leur micro: là oui, ça fonctionne.

Re-retour à mon domicile et réinsertion de ladite carte. Re-RIEN!. Bon, un petit coup de fil; réponse:
- avez-vous regardé les switches (interrupteurs pour les puristes) sur la carte mère?
- Ah bon, il y a des contacteurs (switches pour les franglistes impénitents)? Eh bien pas chez moi!
- Avez-vous bien regardé?
- Ben oui, partout.

Le désespoir me guette. Aurait-je le seul compatible non-compatible par absence de commutateurs (autre mot pour switches)?

Si je vous écrit tout ça, ce n'est pas pour remplir une page, mais vous mettre dans l'ambiance de la démarche de quelqu'un (vous un jour, si ce n'a déja été le cas) qui doute, cherche et trouve. Car certains parmis vous croient encore que nous sommes des génies, alors que nous avons (je crois...) deux bras, deux jambes, une tête avec tout ce qu'il faut dedans pour essayer de résoudre nos problèmes quotidiens et les autres.

Fin de la DIX-GRAISSE-SCIONS (comme le dit Bérurier, dans les livres de SAN-A, ed F.NOIR).

En principe, un compatible PC est équipé sur la carte mère d'une batterie de micro-contacteurs dont le rôle est d'informer le système lors de la mise en route sur la configuration du système. Chaque interrupteur correspond à une sélection particulière au sein d'une configuration globale de type binaire. Ceux concernant le type de contrôleur vidéo sont appelés b2 b3:

```
b2 b3 = 1 1     carte mono-chrome
b2 b3 = x x     pour x x = 01 10 ou 00, carte couleur
```

La configuration est mémorisée à l'adresse 0410:0h. A l'aide de TURBO-Forth, tapez:

```
HEX 0410 0 L@ U. DECIMAL
```

et en principe, vous obtenez la valeur 407Dh. En modifiant cette valeur sur les seuls bits b2,b3, on peut commuter l'affichage d'une carte mono-chrome à une carte couleur et inversement. C'est ce que nous allons appliquer:

LISTING:

```
HEX
0 0410 2CONSTANT SWITCHES
SWITCHES L@ CONSTANT CONFIG-INITIALE
: COLOR-CARD ( ---)
   SWITCHES L@
       CF AND
       20 OR
   SWITCHES L!
   3 MODE \provoque effacement
   ; \ avec sélection mode couleur

: MONO-CARD
   CONFIG-INITIALE SWITCHES L!
   2 MODE ;  \ sélection mode monochrome
DECIMAL
```

Dur de faire plus court! Premier avantage pratique: les deux moniteurs peuvent rester connectés. Vous passez de l'un à l'autre en tapant COLOR-CARD ou MONO-CARD. Cependant, la commutation efface l'écran vidéo activé.

On peut cependant écrire dans un écran tout en conservant la main sur l'autre écran vidéo:

```
HEX
B000 0 CONSTANT MEM-MONO \ orig mémoire monochrome
: >MONO  ( car offset ---)
>R MEM-MONO R> + LC! ;
DECIMAL
```

La même routine peut être utilisée avec une modification mineure pour obtenir le même effet sur carte couleur en 2 MODE:

```
HEX
B800 0 CONSTANT MEM-COLOR \ orig mém couleur
: >COLOR  ( car offset ---)
>R MEM-COLOR R> + LC! ;
DECIMAL
```

Exemple, passons sur le moniteur couleur:

```
COLOR-CARD
```

L'affichage du caractère A sur le moniteur monochrome peut être provoqué par:

```
65 0 >MONO
```

ce qui affiche A à la ligne 0, colonne 0 du moniteur monochrome.

Voilà, je vous fait confiance maintenant pour tripoter EMIT, KEY et CR et faire des choses étonnantes avec ces premiers éléments.

FORTH

DES NOUVELLES DE TURBO-Forth

Maintenant des nouvelles de TURBO-Forth: ceux qui ont reçu une version datée d'avant le 25/01/88 feraient bien d'en commander une nouvelle, car la version actuelle et quasi-presque-intégralement complète contient des routines dont nous ferons mention mais ne figurant pas sur leur version:

- STATUS a été activé sur le prompt OK et sélectable par STATUS? ON ou OFF. Si vous tapez STATUS? ON, l'affichage indique:

```
0 (dec) OK
```

avec OK en surbrillance. Le premier chiffre indique le nombre d'éléments présents sur la pile de données. Le message (dec) vous signale que vous êtes en base décimale:

```
(dec)  pour base décimale
(hex)  pour base hexadécimale
(oct)  pour base binaire
```

Le nombre d'éléments disponibles sur la pile reste en décimal même si la base numérique ne l'est pas. Exemple:

```
1 5 15   2 BASE !   et RETURN
```

affiche

```
3 (#02) OK
```

ce qui signifie que vous avez trois nombres sur la pile et que la base numérique courante est binaire.

- ECHO ON ou OFF est une facilité permettant de voir défiler ou non le programme en cours de compilation depuis un fichier texte. Cette commande peut être intégrée au programme pour faire exécuter des listages partiels. Cette option est disponible depuis les premières versions.

- EOF permet un arrêt de compilation dans un fichier. Ce qui suit cette commande peut être un texte commantant le fonctionnement du progamme. J'encourage ceux qui voudraient nous envoyer des listings à profiter de cette facilité. Exemple:

```
\ petit programme
: BC 256 32 DO I EMIT LOOP ;
EOF   \ Explications
Ce programme affiche tous les caractères de code
ASCII compris entre 32 et 255.
```

A la compilation, l'interpréteur s'interrompt à EOF. En cas de fichiers imbriqués, EOF renvoie au fichier appelant.

Le mot EOF peut également vous permettre de voir ce qui ne va pas en cours de compilation en plaçant ce mot à l'endroit délicat. Attention, il n'est pas immédiat, donc ne le placez dans une définition que si vous souhaitez intégrer EOF à la définition de ce mot.

- Le mot DEBUG positionne une variable <IP qui a été déplacée et dont le contenu est analysé. Lors de l'exécution de WORDS ou de SEE, le mot pointé par DEBUG apparaîtra en surbrillance. Essayez:

```
DEBUG CR
SEE WORDS
```

Attention, option disponible seulement depuis le 25/01/88.

- le mot SEE décompile maintenant en indiquant les branchements par numéros. Exemple:

```
: BOUCLE 10000 0 DO LOOP ;
SEE BOUCLE
```

affiche

```
     : BOUCLE
n°1  (LIT) 10000 0 (DO) n°8
n°6  (LOOP) n°6
n°8  ;
```

L'indication n°n est à compter pour un mot, sauf pour celle située en début de ligne.

- le mot ['] a été modifié. Il ne compile plus une valeur litérale, ce qui rend la décompilation beaucoup plus explicite.

- les touches de fonctions sont gérées directement par FORTH et non plus le driver ANSI.SYS. Ainsi, en sortie de TURBO, les touches précédemment affectées par un autre logiciel ne sont pas perturbées. Par contre, les attributs d'affichage sont toujours gérés par le driver ANSI.SYS. Si l'affichage des attributs vous gène, tapez ATTRIBUTS OFF.

- Un nouveau mot, LSAVE, permet la sauvegarde de zones mémoire extra-segment de 64k maximum. Le mot SAVE utilise dorénavant LSAVE. Rien de changé concernant la syntaxe de SAVE, sauf qu'en cas d'oubli de l'extension, c'est .COM qui est automatiquement rajouté.

- Les erreurs en cas de compilation sont gérées par rappel de l'éditeur et positionnement en ligne et colonne où l'erreur a été détectée. Vous pouvez utiliser l'éditeur depuis le DOS en tapant:

```
EDIT fichier.ext lig col
```

où lig et col sont les valeurs numériques des lignes et colonnes où positionner le curseur. En cas d'oubli de ces valeurs, vous êtes positionnés par défaut sur la première ligne. Si vous ne précisez pas le nom du fichier, c'est le nom SANSNOM.TXT qui est pris par défaut (on considère que TOTO est quasiment un mot réservé du système tellement il est employé à titre d'exemple par les débutants).

L'option ^K^D sauvegarde le fichier avec le même nom que celui de chargement et vous renvoie sous FORTH (commande similaire à celle de WORDSTAR).

Rappelons encore une fois que si notre éditeur ne vous convienne pas, utilisez la commande:

```
editeur.COM EDIT$ $!
```

où editeur.COM est le nom de votre programme de traitement de texte. L'appui sur F2 appellera votre éditeur.

FORTH

MENUS DEROULANTS

par Marc PETREMANN

pour TURBO-Forth exclusivement

On me demande parfois à quoi peut bien servir FORTH en dehors des applications orientées sur le langage FORTH. Voici donc le premier programme inter-logiciel proposé dans les pages de JEDI.

Ce programme permet de gérer un menu déroulant. Le résultat est passé de deux manières au programme appelant:

- par BYE si le programme appelant est un fichier .BAT
- par génération d'une ligne de syntaxe dBASE dans un fichier ASCII CHOIX.PRG si le programme appelant est dBASE III ou dBASE III+.

Une option SAUVE-VIDEO et RESTAURE-VIDEO permet de sauvegarder le contenu de l'affichage avant lancement du menu déroulant.

Plusieurs menus déroulants peuvent être compilés dans un même programme. Voir l'exemple en fin de listing.

LISTING:

```
\ Sauvegarde et restitution écran vidéo
VARIABLE SCR#1 4000 ALLOT  \ variable de sauvegarde
HEX B800 0 2CONSTANT ORIG-VIDEO \ carte CGA
  \ B000 0 2CONSTANT ORIG-VIDEO si carte monochrome
 DECIMAL
DSEGMENT SCR#1 2CONSTANT DEST-VIDEO
: SAUVE-VIDEO ( ---)
     ORIG-VIDEO DEST-VIDEO 4000 LCMOVE ;
: RESTAURE-VIDEO ( ---)
     DEST-VIDEO ORIG-VIDEO 4000 LCMOVE ;

:: DEFINED NIP IF [COMPILE] \ THEN ; ALEPH    INCLUDE
ALEPH.FTH
\ charge ALEPH.FTH si ALEPH n'est pas défini
\ ALEPH.TXT contient la gestion de variables locales,
\ programme diffusé dans JEDI 39, page 2 listing 1.

<DECLARE   VAR XD   VAR YD   VAR XF   VAR YF
<DEFINE    DOUBLE ( xd yd xf yf ---)
    XD YD  AT ASCII ╔ EMIT    XF XD 1+
    DO   I YD AT ASCII ═ EMIT LOOP
    XF YD  AT ASCII ╗ EMIT    YF YD 1+
    DO   XD I AT  ASCII ║ EMIT  XF I AT  ASCII ║ EMIT LOOP
    XD YF  AT ASCII ╚ EMIT    XF XD 1+
    DO   I YF AT ASCII ═ EMIT LOOP
    XF YF  AT ASCII ╝ EMIT           >>

VARIABLE CHOIX      \ numéro du choix courant
VARIABLE #CHOIX     \ nombre de choix maximum
: TITRE
    CREATE SWAP  C,  C,     ( compilation position x et y
d'affichage)
        DUP C,        ( compilation longueur de chaîne)
```

```
        HERE >R         ( sauvegarde de DP)
        DUP ALLOT       ( réservation place pour chaîne)
        R>              ( récupération de HERE)
        SWAP CMOVE      ( compilation de la chaîne)
    DOES>
        DUP C@
        OVER 1+ C@ AT ( positionne le curseur en x y)
        2+ COUNT TYPE ATTOFF ;

DEFER SELECTION

: ROULEAU ( ---)
    0 CHOIX !   0 INVERS SELECTION ATTOFF
    #CHOIX @ 1 ?DO  I SELECTION LOOP
    BEGIN
        KEY   DUP  CHOIX @ SELECTION
        CASE ASCII + OF  1 CHOIX +!  ENDOF
             ASCII - OF -1 CHOIX +!  ENDOF
    ENDCASE
    CHOIX @ 0 MAX  #CHOIX @ 1 - MIN  DUP CHOIX !
    INVERS SELECTION ATTOFF
    13 = UNTIL
    ;

: TO-BATCH ( ---)
    ROULEAU
    CHOIX @ BYE ;

255 STRING CHOIX$
: TO-DBASE ( ---)
    ROULEAU
    " " CHOIX$ $!
    " STORE " CHOIX$ APPEND$
    CHOIX @ (.) CHOIX$ APPEND$
    "  TO CHOIX  " CHOIX$ APPEND$
    CONTROL Z CHOIX$ 1- + C!
    " DSEGMENT CHOIX$ OVER + SAVE CHOIX.PRG" $EXECUTE  ;

EOF
```

Exemple d'utilisation de menu déroulant

1) Syntaxe de titre:
"chaine" N°option colonne ligne TITRE <nom>

2) initialisation des rubriques:

```
  3 #CHOIX !
  " RECAPITULATIF MENSUEL        "  10 10 TITRE RECAP1
  " SITUATION DES CONSOMMATIONS  "  10 11 TITRE RECAP2
  " ESTIMATIONS KILOMETRIQUES    "  10 12 TITRE RECAP3

  #CHOIX @ CASE: CARBUR
    RECAP1 RECAP2 RECAP3 ;
```

3) ensuite choisir pour une utilisation à partir d'un fichier .BAT:

```
: MENU1
    3 #CHOIX !
    ['] CARBUR IS SELECTION
    TO-BAT  ;
```

(Cet exemple est réellement utilisé dans une application professionnelle. Je ne vous raconte pas la tête des collègues programmeurs quand ils ont su que c'était programmé en FORTH...).

Pour une utilisation à partir de dBASE III ou dBASE III+:

```
: MENU2
    3 #CHOIX !
    ['] CARBUR IS SELECTION
    TO-DBASE ;
```

Faire une sauvegarde sous forme compilée par SAVE-SYSTEM:

```
SAVE-SYSTEM CARBUR.COM
```

4) Utilisation pratique:

A partir d'un fichier .BAT, on utilisera le programme en tapant

```
ECHO lancement de CARBUR
CARBUR MENU1
ON ERRORLEVEL 2 GOTO choix2
ON ERRORLEVEL 1 GOTO choix1
ON ERRORLEVEL 0 GOTO choix0
```

Et à partir d'un programme dBASE III ou dBASE III+

```
RUN CARBUR MENU2
DO CHOIX.PRG
* préciser l'extension et déclarer éventuellement PUBLIC CHOIX
* pour la variable CHOIX
```

Le mot TO-DBASE génère un fichier ASCII d'une ligne nommé CHOIX.PRG de syntaxe dBASE et contenant une ligne:

```
STORE n TO choix
```

où n est la valeur sélectionnée dans le menu déroulant.

Nota: avec des fichiers très volumineux, j'ai eu quelques problèmes d'exécution du menu déroulant (plantage) quand plusieurs menus sont compilés dans un même fichier et que tout le fichier fait plus de 32k.

LISP

UN ROBOT LOGICIEL

par A. JACCOMARD.

Le-Lisp MSDOS

Ce programme a été écrit avec l'interprète Le_Lisp, version 15.2, pour MS-DOS. L'essentiel en a été repris du livre de Ch. QUEINNEC "Langage d'un autre type : LISP" (cf. références en fin d'article).

Le robot possède un bras lui permettant de manipuler des objets, un oeil pour les voir. Son univers est constitué des objets fournis par l'utilisateur, ces objets ayant un type, une forme et une couleur donnés. Il comprend les ordres, du moins s'ils sont réalisables dans son univers: il raisonne, dans des limites somme toute assez étroites.

Voici un exemple de dialogue que l'on peut mener avec le robot :

```
? (robot)
   BONJOUR, à vos ordres !
  Ordonne, Maitre
--> (je te donne une boite bleue nommée alpha)
 Merci pour  la boite (alpha) bleue.
 Ordonne, Maitre

--> (et puis voici une boite rouge appelée beta)
 Attendez  que je me libère la main ...
 C'est fait.  J'ai déposé  la boite  (alpha) bleue sur le sol.
 Merci pour la boite (beta) rouge.
 Ordonne, Maitre

--> (maintenant  voici une sphère rouge dénommée gamma)
 Attendez que je me libère la main ...
 C'est fait. J'ai déposé la  boite  (beta)  rouge  sur le sol.  Merci pour la sphère (gamma) rouge.

--> (que vois-tu ?)
 Je regarde la sphère (gamma) rouge.
  Ordonne, Maitre

--> (prend alpha dans ta main)
 Mais attendez que je pose la sphère (gamma) rouge sur le sol.
 C'est fait. J'ai déposé la sphère  (gamma) rouge  sur le sol.
 C'est fait. J'ai pris la boite (alpha) bleue en main.
    Ordonne, Maitre

--> (pose alpha sur beta)
```

C'est fait. J'ai posé la boite (alpha) bleue sur la boite (beta) rouge.
Ordonne, Maitre

--> (et enfin je te donne une pyramide argentée dénommée delta)
Merci pour la pyramide (delta) argentée.
Ordonne, Maitre

--> (dépose delta s'il te plait)
C'est fait. J'ai déposé la pyramide (delta) argentée sur le sol.
Ordonne, Maitre

--> (que sais-tu ?)
Je sais que :
Je regarde la pyramide (delta) argentée.
Je ne tiens rien.
Il y a une pile composée de : la pyramide (delta) argentée
et c'est tout pour cette pile.
Il y a une pile composée de : la sphère (gamma) rouge
et c'est tout pour cette pile.
Il y a une pile composée de : la boite (beta) rouge
avec au-dessus la boite (alpha) bleue
et c'est tout pour cette pile.
Il n'y a rien d'autre.
Ordonne, Maitre

--> (bon pose gamma sur alpha)
Attendez que je prenne en main la sphère (gamma) rouge...
C'est fait. J'ai pris la sphère (gamma) rouge en main.
C'est fait. J'ai posé la sphère (gamma) rouge sur la boite (alpha) bleue.
Ordonne, Maitre

--> (plus délicat maintenant pose beta sur alpha)
Attendez que je prenne en main la boite (beta) rouge .
Mais attendez que je dégage la boite (beta) rouge.
Il y a la boite (alpha) bleue dessus.
Mais attendez que je dégage la boite (alpha) bleue .
Il y a la sphère (gamma) rouge dessus.
C'est fait. J'ai pris la sphère (gamma) rouge en main.
C'est fait. J'ai déposé la sphère (gamma) rouge sur le sol.
C'est fait. J'ai pris la boite (alpha) bleue en main.
C'est fait. J'ai déposé la boite (alpha) bleue sur le sol.
C'est fait. J'ai pris la boite (beta) rouge en main.
C'est fait. J'ai posé la boite (beta) rouge sur la boite (alpha) bleue.
Ordonne, Maitre

--> (donne-moi delta)
Attendez que je retrouve la pyramide (delta) argentée ...
C'est fait. J'ai pris la pyramide (delta) argentée en main.
Voilà, je vous rends la pyramide (delta) argentée.
Ordonne, Maitre

--> (que sais-tu ?)
Je sais que :
Je ne vois rien de particulier.
Je ne tiens rien.
Il y a une pile composée de :
la boite (alpha) bleue
avec au-dessus la boite (beta) rouge
et c'est tout pour cette pile.
Il y a une pile composée de :
la sphère (gamma) rouge
et c'est tout pour cette pile.
Il n'y a rien d'autre.
Ordonne, Maitre

--> (c'est bien arrête-toi)
Au revoir, Maitre.
=?

Le robot possède encore bien d'autres possibilités. Il utilise essentiellement deux fonctions, FILTRE et MICRO-PARSE, une base de données, DATA, contenant les faits de l'univers et les actions demandées dans cet univers, et un ensemble de règles, RULES, contrôlant les interactions entre faits et actions.

La fonction FILTRE prend une expression, un filtre et un environnement (une A-liste) et retourne un environnement, ou NIL en cas d'insuccès. Les filtres sont au nombre de trois :

- (*ONE*) qui filtre un objet quelconque;
- (*ONE* identificateur) qui de plus associe la valeur filtrée à la variable 'identificateur'. Cette variable sera éventuellement réutilisée pour filtrer une même valeur;
- (*ANY*) filtre une suite quelconque d'objets quelconques;
- (*LISTE* X Y Z ...) filtre une liste contenant un terme filtrable par l'un des filtres X, Y, Z...

MICRO-PARSE traduit en termes d'actions une phrase en langage naturel. Cette fonction prend comme arguments une phrase et une grammaire, et retourne une valeur, synthèse de la phrase d'entrée analysée selon la grammaire. Par exemple, la phrase:

(et maintenant je te donne une sphère rouge nommée alpha)

est analysée par la forme

((*ANY*) je te donne (*ONE*) (*ONE* type) (*ONE* color) (*ONE*) (*ONE* X) (*ANY*))

qui va associer 'type' à 'sphère', 'color' à 'rouge' et X à 'alpha', pour synthétiser la clause 'WANT-TO-GIVE alpha'. Notez que la structure de la phrase est imposée par celle de la forme-filtre.

La base de données est un ensemble ordonné de clauses, représentées par des listes dont le premier terme est un mot-clef identifiant le type de la clause. Elles sont de deux sortes:

- les clauses décrivant l'état de l'univers, au nombre de six:

```
(TYPE    identifiant d'objet   forme)
(COLOR   identifiant d'objet   couleur)
(ON      identifiant d'objet   FLOOR)
(ON      identifiant d'objet   identifiant d'objet)
(HAND    identifiant d'objet)
(EYE     identifiant d'objet)
```

Tous les objets de l'univers du robot (ceux donnés par l'utilisateur) doivent avoir un nom unique, l'identifiant d'objet. Les clauses TYPE et COLOR sont auto-explicites, ON situe les objets dans l'espace, HAND et EYE fixent les objets tenus et regardés par le robot.

- les clauses d'actions se subdivisant en:

- clauses de connaissance:
- (WHAT-IS-KNOWN) : le robot décrit ce qu'il sait,
- (WHAT-IS-SEEN) : ce qu'il voit,
- (WHAT-IS-HELD) : ce qu'il tient.

- clauses d'action proprement dites:
- (WANT-TO-GIVE identifiant d'objet forme couleur) : accepter un objet de l'utilisateur;
- (WANT-TO-RETRIEVE identifiant d'objet) : lui en rendre un ;
- (WANT-TO-TAKE identifiant d'objet) : en prendre un dans la main ;
- (WANT-TO-LAY identifiant d'objet FLOOR) : en déposer un sur le sol ;
- (WANT-TO-LAY identifiant d'objet identifiant d'objet) : en déposer un sur un autre ;
- (WANT-TO-SEE identifiant d'objet) : regarder un objet ;
- (WANT-TO-STOP) : s'arrêter ;
- (WANT-TO-OBEY) : attendre un ordre.

DATA, la base de données, est une liste de clauses ne contenant initialement que la clause (WANT-TO-OBEY). Elle se construit au fur at à mesure des besoins, grâce à ses 5 fonctions de manipulation :

- ADD ajoute une clause à la base (clause d'action en tête, clause d'état à la fin);
- DELETE efface la clause fournie en argument, la première éventuellement;
- FORGET efface la première clause de la base, sans distinction;
- ERASE efface une clause spécifiée par son mot-clef;
- UPDATE met à jour ou crée éventuellement une clause spécifiée par son mot-clef.

La fonction RUN-CLAUSES enchaîne les dialogues, en faisant exécuter séquentiellement les clauses d'actions. Si la clause appartient à la P-liste de ACTION, la fonction associée au mot-clef de la clause considérée est lancée (sans argument). Deux clauses sont définies par ce moyen, WANT-TO-STOP et WANT-TO-OBEY, cette dernière exécute un ordre et ajoute la clause d'action correspondante synthétisée par MICRO-PARSE.

Si la clause appartient à la P-liste de RULES, les règles à appliquer sont associées au mot-clef. Les règles sont ainsi structurées :

règle = ((clause d'appel clauses de contexte ...) expressions ...)

'expressions' étant une suite souvent formée de fonctions de manipulation (mais elles peuvent être des expressions LISP quelconques). Par exemple, la règle

```
(((WANT-TO-TAKE (*ONE* X))    ; clause d'appel
  (HAND (*ONE* X) )           ; clause unique de contexte
 (:ANSWER |Mais j'ai déjà | (*ONE* X) | en main|)
 (:FORGET) )
```

signifie que si l'action consiste à prendre un objet (WANT-TO-TAKE (*ONE* X)) et que cet objet est dèjà dans la main du robot, alors ce dernier doit le dire, puis oublier la clause à exécuter pour passer à la suite.

La fonction ANSWER permet au robot de répondre "intelligement" à l'utilisateur. Elle substitue, dans la réponse, à (*ONE* X) la séquence

"article forme de l'objet (identifiant d'objet) couleur"

et elle reconnait qu'un atome est un identifiant d'objet à sa présence sur la P-liste de OBJECT.

Les règles sont appliquées dans l'ordre où elles apparaissent sur la P-liste de RULES: cet ordre n'est pas indifférent. Le robot ne reçoit un objet que si sa main est vide, sinon il dépose sur le sol l'objet tenu. Pour saisir un objet, il vérifie qu'il ne l'a pas déjà en main, que celle-ci est vide, et qu'il n'y a rien au-dessus de l'objet qu'il désire. Pour poser un objet sur un autre ou sur le sol, il vérifie qu'il a l'objet en main, et que celui sur lequel il doit le poser est libre. Il ne peut rendre un objet qu'après l'avoir pris.

Les erreurs sont signalées de façon bien visible, mais elles ne sont pas toutes corrigées. Ainsi, le robot accepte de poser un objet sur un autre qu'il ne possède pas ...

Ce programme, modulaire et structuré, pourrait être amélioré selon 3 directions :

- en ajoutant des règles à la grammaire ("que connais-tu", énonçant les objets connus du robot);
- en introduisant d'autres filtres ((*OR*) reconnaissant un mot dans une liste);
- par la prise en compte des situations "réelles" (interdire de poser une boule sur une pyramide, ...)

De même, des possibilités de Le_Lisp ne sont pas exploitées: le graphisme et le multi-fenêtrage permettraient de "voir" le robot agir dans son coin. Mais ceci est un autre programme ...

Références:

1 - Interprète Le_Lisp de l'INRIA, version 15.2, fournie par ACT Informatique, PARIS.
2 - Ch. QUEINNEC, "Langage d'un autre type : LISP", aux Editions EYROLLES, PARIS, 1983.

```
; LISTING ---------------------- ROBOT ----------------------

    ; Un robot "logiciel" décrit dans Ch. QUEINNEC "Langage d'un autre type : LISP" [1983].
    ; Adaptation pour Le_Lisp v. 15.2 par A. Jaccomard, déc. 1987.

(defvar #:sys-package:colon 'robot)     ; place le ROBOT dans son "package"
(de robotend ()                         ; récupère la place occupée
   (setq #:sys-package:itsoft (delq 'robot #:sys-package:itsoft)) )

; Les utilitaires.

    ; BUILD construit des listes, et place dans l'expression à construire la valeur associée (dans la
    ; A-liste) à l'argument du 'constructeur' *one* ou *=* .

  (dm :build call
      `(:build1 ,(cadr call) ,(if (cddr call) (caddr call))) )

  (de :build1 (L AL)
      (if (consp L)
          (if (consp (car L))
              (let ((fn (get ':build (caar L))))
                   (if fn (funcall fn L AL)
                       (cons (:build1 (car L) AL)
                       (:build1 (cdr L) AL))) )
              (cons (car L) (:build1 (cdr L) AL)) )
   L ) )

  (putprop ':build                               ; place sur la P-liste de BUILD l'identificateur '*=*'
     (lambda (L AL) (cons (eval (cadar L))      ; et sa fonction associée.
                          (:build1 (cdr L) AL)) )
   '*=*)

  (putprop ':build                               ; identificateur spécial '*one*
      (lambda (L AL) (cons (let ((R (assoc (cadar L) AL)))
                                (if R (cdr R) (car L)))
                           (:build1 (cdr L) AL)) )
   '*one*)
```

```
  (de :lprin (L)                              ; affiche en vidéo inversée les articles de 'L'
      (tyattrib #%01110001)
      (if (consp L) (progn (prin (car L) '| |)
                           (:lprin (cdr L)) )
          (terpri) )
      (tyattrib ()) )

(de  :flag   (A  I)                                                                  ;   place  le   drapeau  I
  de  valeur   T  sur   la  A-liste   de  A.         (putprop  A  T  I)  )

; Filtrage.

  (de :filtre (exp filt al)                   ; 'exp' : expression à filtrer
     (if al                                   ; 'filt' : filtre utilisé
        (if (atomp filt)                      ; 'al' : la A-liste où placer le résultat
            (if (eq exp filt) al)
               (let ((one (get ':filtre (car filt))))
                    (if one (funcall one exp filt al)
                       (:filtre-liste exp filt al) ) ) ) ))

  (de :filtre-liste (exp filt al)
      (if al
         (if (consp filt)
             (if (consp (car filt))
                 (let ((one (get ':filtre-liste (caar filt))))
                      (if one (funcall one exp filt al)
                          (if (consp exp)
                              (:filtre-liste (cdr exp) (cdr filt) (:filtre (car exp) (car filt)
                                        al) ) ) ))
                  (if (consp exp)
                      (:filtre-liste (cdr exp) (cdr filt) (:filtre (car exp) (car filt) al) ) ) )
                      (if (eq exp filt) al) ) ) )

   ; Filtre (*one*) ou (*one* X)
   (putprop ':filtre   (lambda (exp filt al)
             (if (null (cdr filt)) al
                (let ((v (assoc (cadr filt) al)))
                      (if v (if (equal exp (cdr v)) al)
                            (cons (cons (cadr filt) exp)  al) ) ) ) )
    '*one*)

   ; Filtre (*any*)
   (putprop ':filtre-liste (lambda (exp filt al)
            (if (null (cdr filt)) al
                (if (consp exp)
                    (let ((r (:filtre-liste exp (cdr filt) al)))
                          (if r r (:filtre-liste (cdr exp) filt al)))
                    (:filtre-liste exp (cdr filt) al) ) ) )
    '*any*)

   ; Filtre (*liste* X Y Z ...)
   (putprop ':filtre (lambda (exp filt al)
            (if (cdr filt)
                (:filtre exp (cons '*liste* (cddr filt))
                            (:filtre-liste exp `((*any*) ,(cadr filt) (*any*))
                      al) )
              al) )
    '*liste*)

   ; Interface langage naturel.
   (de :micro-parse (exp grammar)                          ; micro-analyse de la phrase entrée
     (if grammar
        (let ((r (:filtre exp (caar grammar) t)))
              (if r (:build (cadar grammar) r)
                  (:micro-parse exp (cdr grammar)) ) ) ) )

   (setqq :robot-grammar (
   (((*any*) je te donne (*one*) (*one* type) (*one* color) ; acquisition d'un objet.
      (*one*) (*one* x) (*any*) )
     (want-to-give (*one* x) (*one* type) (*one* color)) )
   (((*any*) voici (*one*) (*one* type) (*one* color)      ; synonyme exact de "je te donne"
      (*one*) (*one* x) (*any*) )
     (want-to-give (*one* x) (*one* type) (*one* color)) )
   (((*any*) donne-moi (*one* x) (*any*))                  ; rend un objet.
     (want-to-retrieve (*one* x)) )
   (((*any*) prend (*one* x) (*any*))                      ; prend un objet.
     (want-to-take (*one* x)) )
   (((*any*) dépose (*one* x) (*any*))                     ; pose un objet sur le sol.
     (want-to-lay (*one* x) floor) )
   (((*any*) pose (*one* x) sur (*one* y) (*any*))         ; pose un objet sur un autre.
     (want-to-lay (*one* x) (*one* y)) )
   (((*any*) regarde (*one* x) (*any*))                    ; regarde un objet.
```

```
      (want-to-see (*one* x)) )
   (((*any*) que sais-tu (*any*))                         ; examen de l'univers du robot.
     (what-is-known) )
   (((*any*) que sais tu (*any*))                         ; synonyme de "que sais-tu"
     (what-is-known) )
   (((*any*) que vois-tu (*any*))                         ; quel objet est regardé ?
     (what-is-seen) )
   (((*any*) que vois tu (*any*))                         ; synonyme de "que vois-tu"
     (what-is-seen) )
   (((*any*) que tiens-tu (*any*))                        ; quel objet dans la main du robot ?
     (what-is-held) )
   (((*any*) arrête (*any*))                              ; ou "stop" ou "arrête-toi"
     (want-to-stop) )
   (((*any*) stop (*any*))
     (want-to-stop) )
   (((*any*) arrête-toi (*any*))
     (want-to-stop) ) ))

 ;  Fonctions de manipulation de la base de données.

 (de :e-erase (clause)                        ; efface une clause spécifiée par son seul mot-clef
  (setq data
        (letn auxerase ((clauses data))
              (if (cdr clauses)
                  (if (equal (caadr clauses) (car clause))
                      (progn (rplacd clauses (cddr clauses)) data)
                             (auxerase (cdr clauses)) )
                      data ) ) ))

 (de :e-update (clause)                       ; met à jour ou crée une clause spécifiée par
  (setq data                                  ; son mot-clef
        (letn updaux ((clauses data))
              (if (cdr clauses)
                  (if (equal (caadr clauses) (car clause))
                      (progn (rplacd (cadr clauses) (cdr clause)) data)
                             (updaux (cdr clauses)) )
                      (:e-add clause) ) ) ) )

 (de :forget ()                               ; efface la première clause de la base
   (setq data (cons 'data (cddr data))) )

 (de :e-delete (clause)                       ; efface de la base la clause fournie en argument,
  (setq data                                  ; sans distinction.
        (letn auxdel ((clauses data))
              (if (cdr clauses)
                  (if (equal (cadr clauses) clause)
                      (progn (rplacd  clauses (cddr clauses)) data)
                             (auxdel (cdr clauses)) )
                      (:lprin `(**** |DELETE| ,clause |WITH DATA IMPOSSIBLE|))
              (list 'data) ) ) ))

 (de :e-add (clause)                          ; ajoute une clause à la base de données.
   (setq data
         (if (or (get ':rules (car clause)) (get ':action (car clause)))
             (rplacd data (cons clause (cdr data)))
             (nconc data  (list clause)) ) ) )

  ; Les formes FEXPR des fonctions précédentes simplifient la construction
  ; de la base de données.

 (df :erase (args)
     (:e-erase args))

 (df :update (args)
     (:e-update args))

 (df :delete (args)
     (:e-delete args))

 (df :add (args)
    (:e-add  args))

;  ANSWER donne les réponses du robot.

 (df :answer phrase
     (letn auxansw ((phrase phrase))
           (if phrase
              (if (get 'objet (car phrase))
                  (progn (let ((r (:filtre data
                                   (:build '(*liste*
                                           (type  (*=* (car phrase)) (*one* t))
                                           (color (*=* (car phrase)) (*one* c)) ))
                                           t )))
```

```
                                        (if r
                                            (progn  (prin  (get 'article (cdr (assoc 't r))) '| |
                                                           (cdr (assoc 't r)) '| |
                                                           (list (car phrase)) '| |
                                                           (cdr (assoc 'c r)) '| |))
                                                    (terpri)
                                            (:lprin `(**** |OBJECT| ,(car phrase) |INCORRECT IN DATA|)) ))
                                            (auxansw (cdr phrase)) )
                  (if (eq (car phrase) 'floor)
                      (progn (auxansw '(le sol)) (auxansw (cdr phrase)))
                      (prin (car phrase) '| |)
                      (auxansw (cdr phrase)) ) ) ) )
                      (terpri) )

; Pour accorder l'article à l'objet :

 (plist 'article '(cube le sphère la pyramide la boite la))
 (addprop 'article 'la 'boule)                ; extension facile.

;  RUN-CLAUSES  enchaîne les traitements.

 (de :run-clauses ()
     (until  (or (null (cdr data)) (eq (caadr data) 'want-to-stop1))
             (let ((rules (get ':rules (caadr data))))
                  (if rules
                      (do ((rules rules (cdr rules)))
                          ((if rules (:apply-rule (car rules))
                               (:lprin `(**** |NO RULE FOR| ,(cadr data) |IN DATA|))
                               (:forget)
                               (:forget) t )) )
                       (let ((action (get ':action (caadr data))))
                            (if action (funcall action)
                                (:lprin `(**** |NO MEANING ASSOCIATED TO| ,(cadr data) |IN DATA|))
                                (:forget))))) )

 (de :apply-rule (rule)
     (let ((r (:filtre (cadr data) (caar rule) t)))
          (if r
              (let ((rr (:filtre (cddr data) (cons '*liste* (cdar rule) ) r )))
                   (if rr
                       (progn (eprogn (:build (cdr rule) rr)) t ))))) )

; La fonction utilisateur :

 (de robot ()
    (setq #:sys-package:itsoft (cons 'robot #:sys-package:itsoft))
    (setq data (list 'data))                   ; initialise DATA
    (:add (want-to-obey))
    (prompt "--> ")
    (setq at (tyattrib))                       ; sauvegarde mode vidéo courant
    (tycls) (tycursor 0 0)
    (tyattrib #%00001111)                      ; vidéo haute intensité
    (print "                BONJOUR, à vos ordres !")
    (tyattrib at)
    (:run-clauses)                             ; la boucle principale.
    (prompt "? ") )

; Les actions sont placées sur la P-liste de ACTION.

 (putprop ':action  (lambda ()                 ; la règle WANT-TO-STOP, arrête le programme.
        (tyattrib #%00001111) (terpri)
        (:answer |      Au revoir, MAITRE|)
        (terpri)
        (tyattrib at)
        (:forget)
        (setq data (cons 'data (cons '(want-to-stop1) (cdr data)))) )
  'want-to-stop)

 (putprop ':action  (lambda ()                 ; la règle WANT-TO-OBEY, exécute un ordre.
        (terpri)
        (tyattrib #%00001111)
        (print "           Ordonne, MAITRE")
        (tyattrib at)
        (terpri)
        (let ((clause (:micro-parse (read) :robot-grammar)))
              (if clause (progn (:e-add clause)
                                (if (eq (car clause) 'want-to-give)
                                    (:flag 'objet (cadr clause)) ) )
                   (tyattrib #%10001111)        ; clignotant brillant
                   (print "Je n'ai pas compris")
                   (tyattrib at) ) ) )          ; vidéo normale
  'want-to-obey)

; Les règles sont placées sur la P-liste de RULES.
```

```
(mapc (lambda (R) (putprop ':rules (cdr R) (car R)) )
  '( (want-to-give                                  ; le mot-clef précédant chaque règle.
   (((want-to-give (*one* x) (*one* t) (*one* c))
     (hand (*one* y)) )
    (:add (want-to-lay (*one* y) floor))
    (:answer |Attendez que je me libère la main ...|) )
   (((want-to-give (*one* x) (*one* t) (*one* c)))
    (:forget)
    (:add (type (*one* x) (*one* t)))
    (:add (color (*one* x) (*one* c)))
    (:add (hand (*one* x)))
    (:update (eye (*one* x)))
    (:answer |Merci pour| (*one* x) |.|) )
       )                    ; fin du "want-to-give"

 (want-to-retrieve
   (((want-to-retrieve (*one* x))
     (hand (*one* x))
     (type (*one* x) (*one* t))
     (color (*one* x) (*one* c)) )
    (:answer |Voilà, je vous rends| (*one* x) |.|)
    (:erase (eye))
    (:forget)
    (:delete (hand (*one* x)))
    (:delete (type (*one* x) (*one* t)))
    (:delete (color (*one* x) (*one* c))) )
   (((want-to-retrieve (*one* x)))
    (:add (want-to-take (*one* x)))
    (:answer |Attendez que je retrouve| (*one* x) | ...|) )
       )                    ; fin du "want-to-retrieve

 (want-to-take
   (((want-to-take (*one* x))
     (hand (*one* x)) )
    (:forget)
    (:answer |Mais j'ai déjà| (*one* x) |en main.|) )
   (((want-to-take (*one* x))
     (hand (*one* y)) )
    (:add (want-to-lay (*one* y) floor))
    (:answer |Mais attendez que je pose| (*one* y) | ...|) )
   (((want-to-take (*one* x))
     (on (*one* y) (*one* x)) )
    (:add (want-to-lay (*one* y) floor))
    (:add (want-to-take (*one* y)))
    (:answer |Mais attendez que je dégage| (*one* x) |,|)
    (:answer il y a (*one* y) |dessus ...|) )
   (((want-to-take (*one* x))
     (on (*one* x) (*one* y)) )
    (:forget)
    (:delete (on (*one* x) (*one* y)))
    (:answer |C'est fait. J'ai pris| (*one* x) |en main.|)
    (:update (eye (*one* x)))
    (:add (hand (*one* x))) )
       )                    ; fin du "want-to-take"

(want-to-lay
  (((want-to-lay (*one* x) (*one* x)))
   (:forget)
   (:update (eye (*one* x)))
   (tyattrib #%00001111)
   (:answer |Cà va pas, non ? Poser un objet sur lui-même !!!|)
   (tyattrib at) )
  (((want-to-lay (*one* x) (*one* y))
    (on (*one* x) (*one* y)) )
   (:answer Mais (*one* x) est déjà sur (*one* y) | !|)
   (:forget) )
 (((want-to-lay (*one* x) floor)
   (hand (*one* x)) )
  (:forget)
  (:update (eye (*one* x)))
  (:delete (hand (*one* x)))
  (:add (on (*one* x) floor))
  (:answer |C'est fait. J'ai déposé| (*one* x) |sur le sol.|) )
 (((want-to-lay (*one* x) (*one* y))
   (hand  (*one* x))
   (on (*one* z) (*one* y)) )
  (:add (want-to-lay (*one* z) floor))
  (:answer |Attendez que je dégage| (*one* y) de (*one* z) |...|) )
 (((want-to-lay (*one* x) (*one* y))
   (hand (*one* x)) )
  (:forget)
  (:update (eye (*one* x)))
```

```
    (:delete (hand (*one* x)))
    (:add (on (*one* x) (*one* y)))
    (:answer |C'est fait. J'ai posé| (*one* x) sur (*one* y) |.|) )
   (((want-to-lay (*one* x) (*one* y))
     (hand (*one* z)) )
    (:add (want-to-lay (*one* z) floor))
    (:answer |Je pose| (*one* z) |qui me gêne.|) )
   (((want-to-lay (*one* x) (*one* y)))
    (:answer |Attendez que je prenne en main| (*one* x) |...|)
    (:add (want-to-take (*one* x))) )
          )                ; fin du "want-to-lay"

  (what-is-held  (((what-is-held)
     (hand (*one* x)) )
    (:forget)
    (:answer |Je tiens| (*one* x) |.|) )
   (((what-is-held))
    (:forget)
    (:answer |Je ne tiens rien.|) )
          )                ; fin du "what-is-held"

  (what-is-known
   (((what-is-known))
    (:forget)
    (:answer |Je sais que :|)
    (:add (what-is-known1))
    (mapc (lambda (clause)
            (let ((r (:filtre clause '(on (*one* x) floor) t)))
               (if r (:e-add (list 'what-is-known1 (cdr (assoc 'x r))))) ) )
          data)
    (:add (what-is-held))
    (:add (what-is-seen)) )
          )                ; fin du "what-is-known"

  (what-is-known1
   (((what-is-known1 (*one* x)))
     (:forget)
    (:add   (what-is-known2 (*one* x)))
    (:answer |Il y a une pile composée de :| (*one* x)) )
   (((what-is-known1 ))
    (:forget)
    (:answer |Il n'y a rien d'autre.|) )
          )                ; fin du "what-is-known1"

  (what-is-known2
   (((what-is-known2 (*one* x))
     (on (*one* y) (*one* x)) )
    (:update (what-is-known2 (*one* y)))
    (:answer avec au-dessus (*one* y)) )
   (((what-is-known2 (*one* x)) )
    (:forget)   (:answer |et c'est tout pour cette pile.|) )
          )                ; fin du "what-is-known2"

  (what-is-seen
   (((what-is-seen)
     (eye (*one* x)) )
    (:forget)
    (:answer |Je regarde| (*one* x) |.|))
  (((what-is-seen))
    (:forget)
    (:answer |Je ne vois rien de particulier.|) )
          )                ; fin du "what-is-seen"

  (want-to-see
   (((want-to-see (*one* x)))
    (:update (eye (*one* x)))
    (:forget)
    (:answer |C'est fait. Je regarde| (*one* x) |.|) )
          ) ))             ; fin du "want-to-see" et des règles.

* ------------------------------------------------------------------------- *
```

COURRIER

CONTENU DES REVUES PARLANT DE FORTH

Ne serait-il pas possible de présenter le contenu des différentes revues consacrées au langage FORTH (ex: photocopie des contents).

Robert PICHON - 78001 VERSAILLES

REPONSE: nous ne recevons pas toutes les revues parlant de FORTH, malheureusement. De plus il existe trois sorte de revues:

- celles consacrées (presque) exclusivement au FORTH: JEDI, FORTH DIMENSION, VIERTE DIMENSION.

suite page 20

SIMULATION DE COMMANDES DE ROBOTS

EN TURBO PROLOG

Le but de ce programme est essentiellement pédagogique. Il montre comment il est possible de faire évoluer une base de connaissances.

Deux robots "tic" et "tac" se trouvent dans un "micro-monde" défini par une base de connaissances initiale (voir exemple dans le programme). Le programme permet d'agir sur ces robots grâce à des ordres tels que *prendre un objet* ou *mettre un objet dans un lieu*.

Quand on ordonne à un robot de prendre un objet il faut que cet objet puisse être pris. De plus, un robot ne peut tenir qu'un objet à la fois.

Si ces conditions sont réunies alors les actions exécutées entraineront des changements dans la base de connaissances : un fait *tient* sera ajouté pendant qu'un fait *peut_être_pris* sera ôté. Au cas où le robot aurait eu à se déplacer pour prendre l'objet son nouveau lieu est consigné et l'ancien est effacé.

La base de connaissances est également réactualisée quand on ordonne à un robot de mettre un objet dans un lieu.

<u>Examinons les clauses un peu plus en détail</u>.

* *va_dans(Robot,Lieu)* sont les clauses utilisées pour ordonner à un Robot de se rendre dans un Lieu défini. Si le Robot s'y trouve déjà, la 1ère clause affiche "*Robot est dans Lieu*". S'il n'y est pas, l'endroit où il se trouvait est supprimé de la base de connaissances, on l'envoie dans le Lieu voulu, d'où le message "*Robot va dans Lieu*" affiché par la 2ème clause puis ce nouveau fait est ajouté à la base.

* *prend(Robot,Objet)*. La 1ère clause vérifie d'abord que l'Objet peut être pris, auquel cas ce fait est enlevé de la base et le fait tient (Robot,Objet) est ajouté; le message "*Robot tient Objet*" est alors affiché. Si l'Objet ne peut être pris c'est qu'un Autre_Robot le détient. La 2ème clause est chargée d'afficher le message "*Autre_Robot tient déjà Objet*".

* *cherche(Robot,Objet)*. Là également la 1ère clause vérifie que le Robot ne détient pas un objet (qui peut être celui recherché). Dans ce cas un message indique le lieu où se trouve l'Objet, ordre est donné au Robot de s'y rendre (il peut y être déjà : voir *va_dans*) puis de prendre cet Objet (voir *prend*). Sinon la 2ème clause intervient en envoyant le message "*Impossible Robot tient déjà tel objet*".

* *déplace(Robot,Objet,Lieu)*. Ici <u>quatre cas</u> peuvent se produire :

1. Robot tient déjà Objet. On l'envoie alors dans Lieu pour qu'il y place l'Objet.
2. Robot tient un Autre_Objet. On le débarasse de cet Autre_Objet (avec mise à jour correspondante de la base), on lui ordonne de chercher l'Objet, puis on l'envoie dans Lieu pour y placer l'Objet.
3. Un Autre_Robot tient l'Objet. La 3ème clause intervient alors pour afficher le message "*Objet est déjà pris par Autre_Robot*".
4. Robot ne détient aucun objet. On l'envoie chercher cet Objet puis dans le Lieu pour l'y placer.

* *place(Robot,Objet,Lieu)* est appelée éventuellement par déplace afin d'actualiser la base de connaissances puis d'afficher "*Robot a mis Objet dans Lieu*".

Après le choix de l'option **Run** le programme offre dans une fenêtre un menu permettant d'exécuter les 2 ordres indiqués précédemment, de sauvegarder la base de connaissances ou de quitter.

Avant et après chaque ordre donné à un robot les situations initiale et finale seront affichées dans deux fenêtres contiguës, ce qui permet de bien visualiser l'évolution de la base de connaissances.

Les ordres donnés aux robots sont faits de façon conversationnelle.

Voir exemples <u>encadrés 1, 2 et 3</u>.

<u>*Nota*</u> : la base initiale n'est là qu'à titre d'exemple. Il est bien sûr possible de la modifier et aussi d'y mettre plus de 2 robots.

Jean-Paul POSTEC

```
/*****************************************************************************/
/*  PROGRAMME DE SIMULATION DE COMMANDES DE ROBOTS AVEC EVOLUTION DYNAMIQUE */
/*    D'UNE BASE DE CONNAISSANCES.  AUTEUR : Jean-Paul POSTEC    Avril 1987  */
/*****************************************************************************/

database
   est_dans(symbol,symbol)              /*************************/
   peut_être_pris(symbol)               /* BASE DE CONNAISSANCES */
   tient(symbol,symbol)                 /*************************/

predicates
   fenêtres
   place(symbol,symbol,symbol)
   déplace(symbol,symbol,symbol)
   va_dans(symbol,symbol)
   cherche(symbol,symbol)
   prend(symbol,symbol)
   effacer_base
   afficher_base
   init
   run
   menu(integer)
   action(integer)

goal
   init,run.

clauses

/*****************************************************************************/
/*                  CREATION DES DIFFERENTES FENETRES                        */
/*****************************************************************************/

  fenêtres if makewindow(1,2,3,"Situation initiale",0,0,15,28),
              makewindow(2,2,5,"Situation finale",0,28,15,28),
              makewindow(3,64,7,"MENU",0,56,15,24),
              makewindow(4,31,2,"Dialogues",15,0,10,80).

/*****************************************************************************/
/*      COMMANDES RELATIVES AUX ROBOTS AVEC MISE A JOUR DE LA BASE          */
/*****************************************************************************/

   place(Robot,Objet,Lieu) if retract(tient(Robot,Objet)),
                              asserta(peut_être_pris(Objet)),
                              retract(est_dans(Objet,_)),
                              asserta(est_dans(Objet,Lieu)),
                              write(Robot," a mis ",Objet," dans ",Lieu), nl.

   déplace(Robot,Objet,Lieu) if tient(Robot,Objet), !,
                                write(Robot," tient ",Objet), nl,
                                va_dans(Robot,Lieu),
                                place(Robot,Objet,Lieu).
   déplace(Robot,Objet,Lieu) if tient(Robot,Autre_objet), !,
                                retract(tient(Robot,Autre_objet)),
                                asserta(peut_être_pris(Autre_objet)),
                                write(Robot," se débarrasse de ",Autre_objet),
                                nl, cherche(Robot,Objet),
                                va_dans(Robot,Lieu),
                                place(Robot,Objet,Lieu).
   déplace(_,Objet,_) if tient(Autre_robot,Objet), !,
                         write(Objet," est déjà pris par ",Autre_robot), nl.
   déplace(Robot,Objet,Lieu) if cherche(Robot,Objet),
                                va_dans(Robot,Lieu),
                                place(Robot,Objet,Lieu), !.

    va_dans(Robot,Lieu) if est_dans(Robot,Lieu), !,
                           write(Robot," est dans ",Lieu), nl.
    va_dans(Robot,Lieu) if retract(est_dans(Robot,_)),
                           write(Robot," va dans ",Lieu), nl,
                           asserta(est_dans(Robot,Lieu)).

    cherche(Robot,Objet) if not(tient(Robot,_)), !,
                            est_dans(Objet,Lieu),
                            write(Objet," est dans ",Lieu), nl,
                            va_dans(Robot,Lieu),
                            prend(Robot,Objet).
    cherche(Robot,_) if tient(Robot,Autre_objet),
                        write("Impossible ",Robot," tient déjà ",Autre_objet), nl.
```

```
    prend(Robot,Objet) if peut_être_pris(Objet), !,
                          retract(peut_être_pris(Objet)),
                          write(Robot," prend ",Objet), nl,
                          asserta(tient(Robot,Objet)), !.
    prend(_,Objet) if tient(Autre_robot,Objet),
                   write("Impossible ",Autre_robot," tient déjà ",Objet), nl.

/*****************************************************************************/
/*    SERT A EVITER LE CUMUL DE LA BASE DE CONNAISSANCES EN CAS D'ECHEC     */
/*****************************************************************************/

  effacer_base if est_dans(Robjet,Lieu), retract(est_dans(Robjet,Lieu)), fail.
  effacer_base if peut_être_pris(Objet), retract(peut_être_pris(Objet)), fail.
  effacer_base if tient(Robot,Objet), retract(tient(Robot,Objet)), fail.
  effacer_base.

/*****************************************************************************/
/*       AFFICHAGE DE LA BASE AVANT ET APRES TOUTE COMMANDE DE ROBOT        */
/*****************************************************************************/

  afficher_base if nl, est_dans(Robjet,Lieu),
                   write(Robjet," est dans ",Lieu), nl, fail.
  afficher_base if nl, peut_être_pris(Objet),
                   write(Objet," peut être pris"), nl, fail.
  afficher_base if nl, tient(Robot,Objet),
                   write(Robot," tient ",Objet), nl, fail.
  afficher_base.

/*****************************************************************************/
/*        INITIALISATION AVEC CONSULTATION DE LA BASE SUR DISQUE            */
/*****************************************************************************/

  init if effacer_base, fenêtres, consult("mabase.dba").

/*****************************************************************************/
/*                        DEROULEMENT DU PROGRAMME                          */
/*****************************************************************************/

  run if shiftwindow(1), clearwindow, afficher_base,
         shiftwindow(2), clearwindow, menu(Touche),
         shiftwindow(4), clearwindow, action(Touche),
         shiftwindow(2), clearwindow, afficher_base,
         shiftwindow(4), cursor(5,56), beep, write("APPUYER SUR UNE TOUCHE"),
         cursor(6,60), write("POUR CONTINUER"), readchar(_),
         clearwindow, run.

/*****************************************************************************/
/*                                  MENU                                    */
/*****************************************************************************/

  menu(Choix) if shiftwindow(3), clearwindow,
      write("1.Ordonner à un robot   de prendre un objet"), nl,
      write("2.Ordonner à un robot   de mettre un objet    dans un lieu"),nl,
      write("3.Sauvegarder la si-    tuation actuelle"), nl,
      write("4.Quitter"), nl, nl,
      write("Entrez votre choix "), readint(Choix).

/*****************************************************************************/
/*              ACTIONS A REALISER SELON LE CHOIX DU MENU                   */
/*****************************************************************************/

  action(1) if write("De quel robot s'agit-il ? "), readln(Robot), nl,
               write("Quel objet doit-il prendre ? "),readln(Objet), nl,
               cherche(Robot,Objet), !.
  action(2) if write("De quel robot s'agit-il ? "), readln(Robot), nl,
               write("Quel est l'objet à placer ? "), readln(Objet), nl,
               write("Quel est le lieu de destination ? "), readln(Lieu), nl,
               déplace(Robot,Objet,Lieu), !.
  action(3) if save("mabase.dba"), !.
  action(_) if exit.
```

```
/****************************************************************************/
/*     EXEMPLE DE BASE DE CONNAISSANCES INITIALE SAUVEE SOUS "mabase.dba"  */
/*                                                                          */
/*                LES DEUX ROBOTS S'APPELLENT "tic" et "tac"                */
/****************************************************************************/

est_dans("tic","cuisine")
est_dans("tac","cave")
est_dans("vin","cave")
est_dans("couteau","cuisine")
est_dans("verre","cave")
est_dans("miroir","chambre")

peut_être_pris("verre")
peut_être_pris("vin")
peut_être_pris("miroir")
peut_être_pris("couteau")
```

suite de la page 16

- celles consacrant épisodiquement des articles au FORTH: MICRO-SYSTEMES, SOFT et MICRO, AMSTRAD MAGAZINE (devenu AMSMAG), DrDOBB'S JOURNAL, THEORIC, THEOPHILE (devenu THEO puis TEO), CHIP.

- celles dont les articles font mention de FORTH, de ses applications.

Ah, j'oubiais, il y a des revues qui ignorent superbement FORTH (l'OI, SVM pour exemple). Comme le dit si bien le Pr CHORON: qu'ils cr.....! (là les petits points c'est pour l'ambiguité, celle qui a valu la porte à M.POLAC et la suppression de droit de réponse, l'émission qui faisait la meilleure audience le samedi après 22h30; comme le dit Mr BOUYGUES, on ne change pas une équipe qui gagne... Au fait, qui construit le tunnel sous la Manche?...).

Plus sérieusement: il est très difficile de récolter toutes les références concernant FORTH parues dans toutes les revues. Si quelqu'un a le courage de le faire, on les publiera très certainement.

Concernant les photocopies d'articles, c'est toujours un peu délicat, car il y a le problème du copyright, problème disparaissant avec le titre si la revue n'est plus diffusée (MICRO-STRAD, MICRO-TOM, LIST, etc...). Mais le contenu de ces articles est-il encore d'actualité (FORTH pour ZX81, bof...)?

L'idéal est encore de faire nous-même l'actualité: exemple TURBO-FORTH, FORTHLOG II, fBASE, BLAISE (le langage PASCAL écrit en FORTH), et tout ce que vous ferez de vos dix doigts sur un clavier.

LE SECRETAIRE

PROPOSITION D'ARTICLE:

Si je propose des trucs en FORTH (Unix ou VolksFORTH) sur ATARI ST, aurai-je une audience? Je cherche d'autres membres passionnés comme moi de musique assistée par ordinateur (pourquoi pas en FORTH), à l'aide!!

Mes micros: ACORN ATOM: en Forth
BBC : tous langages
ATARI ST : tous langages

Mes synthés: YAMAHA FB01
CASIO CZ 3000
HYBRITEC MUSIC 500

Je programme sur BBC avec synthés en AMPLE (version FORTH du MCL - Music Complilway Language-) et je cherce actuellement le moyen de faire des images avec le Volks-Forth sur ATARI, mais je n'ai pas de doc sur le sujet. Le Laxen et Perry apporte-t-il des solutions ou existe-t-il autre chose?

Au fait: mon futur micro, ARCHIMEDE, vous savez, le micro le plus rapide et le moins cher du monde (3x plus rapide qu'un DEXPRO 386 et 6x moins cher) le tout peut-être en FORTH...

Michel OSSELIN - 92800 PUTEAUX

REPONSE: Ouf, que d'activité. Oui, les synthés intéressent certainement nos adhérents et vos trucs en Volks-FORTH aussi. Cette version est amenée à se répandre rapidement parmis nos adhérents depuis qu'elle a été francisée.

Dès que vous aurez votre ARCHIMEDE sous la main, tenez-nous au courant de ses capacités logicielles (vitesses, langages, etc...) car la technologie RISC sera certainement prédominante sur les futurs systèmes. D'ailleurs FORTH n'est pas en retard avec sa puce NC 4016 de NOVIX (mettez un VAX dans votre PC...).

Concernant les images, VolksFORTH est certainement plus performant que F83 de Laxen et Perry. Avec un peu de patience (si vous ne trouvez pas d'ici là), il y aura certainement un adhérent pour trouver les réponses que vous vous posez.

LE SECRETAIRE

FELICITATIONS:

Je vous exprime mon très grand plaisir de lire les 2 derniers numéros de JEDI, où il est question de TURBO-Forth qui nous libère des contraites des blocs d'édition, et des fonctions graphiques qui apportent une nouvelle dimension à F83. Vous avez vraiment donné un grand regain d'intérêt à notre estimé langage.

En utilisant le PSET de ces fonctions et en utilisant la métode d'une table de SINUS (FORTH de SALMAN, TISSERAND et TOULOUT, ed Eyrolles, p174), j'ai constaté qu'un cercle se dessine grosso-modo 1,5x plus vite qu'avec QuickBasic de MICROSOFT! Bien sûr, en n'utilisant aussi que PSET et non CIRCLE, dans QuickBasic.

Luan NGUYEN HUU - 75014 PARIS

REPONSE: Merci de votre intérêt pour TURBO-Forth. Mais nous ne nous arrêterons pas là: bientôt il y aura des routines d'auto-documentation pour TURBO-Forth:

```
HELP <mot>   affichera un message d'information.
```

Exemple:

```
HELP DUP   affichera
DUP     n--n n  Duplique contenu du sommet de pile
```

Ceci pour tout le vocabulaire FORTH (plus de 500 mots à auto-documenter, soit plus de 35k de fichier ASCII en accès direct). Encore un plus de JEDI pour TURBO-Forth.